Functions in Python

Jonathan Young
Master’s Postdoctoral Fellow, DBMI
jdy10@pitt.edu

This material is adapted from the MITx course “6.00x Introduction to Computation and Programming Using Python,” by Eric Grimson. www.edx.org

Free High Quality Education

- Massive Open Online Course
  - www.edx.org (Harvard, MIT, Berkeley consortium) 207 courses
  - www.coursera.org 682 courses
  - www.udacity.com 36 courses
  - https://class.stanford.edu 15 courses
  - www.coursetalk.org
Getting help in Python

• `help(function_name)`
  – Python documentation, docstring

• `dir(function_name)`
• `dir(module_name)`
  – List of functions/methods associated with module/function
Functions

• So far, you’ve seen numbers, assignments, , different data types, input/output, conditionals, and looping constructs
• But code lacks abstractions
  – Have to copy a block of code every time you want to use it
  – Can’t use same variable names in other pieces of code
  – Can quickly get cumbersome to read and maintain
• Functions give us abstraction—allow us to capture computation and treat as if built-in.

A simple example

• Suppose we want z to be the maximum of two numbers, x and y
• A simple script would be

```python
x = 19
y = 119
if x > y:
    z = x
else:
    z = y
```
Capturing computation as a function

- Idea is to encapsulate this computation within a scope such that can treat as a built-in
  - Use by simply calling name and providing input
  - Internal details hidden from user (abstraction)
- Syntax

```python
def <function name> (<formal parameters>):
    <function body>
```
- `def` is a keyword
- Name is any legal Python name
- Within parentheses are zero or more formal parameters – each is a variable name to be used inside function body
- Example 1

A simple example

```python
def max(x, y):
    if x > y:
        return x
    else:
        return y
```
- We can invoke with

```python
z = max(3, 4)
```
- When we call or invoke `max(3, 4)`, `x` is bound to 3, `y` is bound to 4, and then the body expression(s) are evaluated. 3 and 4 are called arguments
Function returns

- Body can consist of any number of legal Python expressions (loops, conditionals, assignments, etc.)
- Expressions are evaluated until
  - Run out of expressions, in which case the special value `None` is returned
  - Or until special keyword `return` is reached, in which case subsequent expression is evaluated and that value is returned as the value of the function call

Summary of function call

- Expressions for each parameter are evaluated, bound to formal parameter names of function
- Control transfers to first expression in body of function
- Body expressions executed until `return` keyword reached (returning value of next expression) or run out of expressions (return `None`)
- Invocation is bound to the returned value
- Control transfers to next piece of code in main program
Problem 1

• Go to http://repl.it/
• Select Python (on the right)
• Write a Python function, square, that takes in one number and returns the square of that number

Problem 2

Write a Python function, eval_quadratic(a, b, c, x) that returns the value of the quadratic

\[ a \cdot x^2 + b \cdot x + c \]
Problem 3

• Write a function, `print_name`, that takes in 3 arguments – 2 strings (`first_name`, `last_name`) and a boolean variable (`reverse`), and prints the names in the correct order depending on the value of `reverse`.

• Default values

Video – Environments/Scoping

• Eric Grimson, MIT Computer Scientist
Example 2 – $f(x)$

Some observations

- Each function call creates a new environment, which scopes bindings of formal parameters and values, and of local variables (those created with assignments within body)
Practice Reading Code

• Examples 3, 4, 5, 6

Specifications/Docstring

• Are a contract between implementer of the function and user
  – Assumptions: conditions that must be met by users of function. Typically constraints on parameters, such as type, and sometimes acceptable ranges of values
  – Guarantees: Conditions that must be met by function, provided that it has been called in way that satisfies assumptions
Example

def isVowel(char):
    '''
    char: a single letter of any case
    returns: True if char is a vowel and False otherwise.
    '''

What do functions do

- Can now create new procedures/functions and treat as if Python built-ins
- Properties
  - Decomposition: Break problems into modules that are self-contained, and can be reused in other settings
  - Abstraction: Hide details. User need not know interior details, can just use as if a black box
Using functions in modules

- Modularity suggests grouping functions together that share a common theme
- Place in a single .py file
- Use `import` command to access

Example

```python
pi = 3.14159

def area(radius):
    return pi*(radius**2)

def circumference(radius):
    return 2*pi*radius

def sphere_surface(radius):
    return 4.0*area(radius)

def sphere_volume(radius):
    return (4.0/3.0)*pi*(radius**3)
```

Be sure to save this code in a file called `circle.py` in current working directory.
Final Problem

• Password generator
• Write a function, `password_generator`, that has a single integer parameter \( n \) and returns a random sequence of letters and numbers of length \( n \).
  
  • `import random`
  
  • Search for a function in the random model that seems appropriate

Returning tuples

```python
def add(x, y, z):
    one = x
    two = x + y
    three = x + y + z
    return one, two, three

a, b, c = add(6, 7, 8)
```

• Unpack the tuple
K-means clustering

- Andrew Ng, Stanford Computer Scientist
- Stop at 3 min 40 seconds

Don’t forget to check-out
www.edx.org